

Introduction à Plack

franck cuny

<linkfluence>

Plack

The New Perl WebServer

Pour être plus précis

PSGI

Plack

plackup

Plack::Handler::*

Plack::Middleware::*

Rapide vocabulaire

PSGI

est une **spécification**

Plack

est une **implémentation**

plackup

est un **utilitaire**

Handler

est une **interface**
entre Plack et le serveur

Middleware

est une **interface**
entre l'application
et le serveur web

PSGI

Perl Web Server Gateway Interface

Qu'est ce que PSGI ?

PSGI est une interface entre le serveur web et une application web écrite en Perl, similaire à ce que fait CGI entre le serveur web et un script CGI.

Ce que n'est pas PSGI !

PSGI n'est **PAS** un framework
pour développer des
applications web.

Requête HTTP

Browser => Serveur web =>
Plack/Middleware => App

Réponse HTTP

App => Plack/Middleware =>
Serveur HTTP => Browser

Port de :

Rack (ruby)

WSGI (python)

Prêt pour la production

version 1.0 bientôt

API Stable

Communauté active

composée des auteurs des
frameworks Catalyst Mojo
Dancer Squatting ...

De nombreux framework web en Perl

Catalyst Dancer Mojo Maypole
Slede Spoon Titanium Ark
Squatting Tatsumaki Reaction Jifty
CGI::Application Apache2::REST
WebGUI Sweetpea

...

La plus part supportent
mod_perl et CGI

Certains supportent FCGI

**Certains ont un mode
standalone**

Problème

Duplication de code !

Duplication d'efforts !

Objectif PSGI:

Unifier les interfaces
entre les applications
et les serveurs web

Comprendre ce que fait PSGI

Application PSGI

code reference

```
my $app = sub {...}
```

Seul les développeurs de framework ont à se soucier de comment écrire l'application PSGI

L'utilisateur du framework n'a pas à se soucier de savoir comment et où elle est écrite

```
my $app = sub {  
  my $env = shift;  
  return [  
    200,  
    ['Content-Type' => 'text/html'],  
    ['hello world']  
  ]  
};
```

```
my $app = sub {  
    my $env = shift;  
    return [  
        200,  
        ['Content-Type' => 'text/html'],  
        ['hello world']  
    ]  
};
```

Une hash d'environnement

Même principe que celle de
CGI.pm

avec des informations propres
à PSGI

HTTP_ACCEPT: application/xml,application/xhtml+xml,text/html

HTTP_ACCEPT_CHARSET: UTF-8,*;q=0.5

HTTP_ACCEPT_ENCODING: gzip,deflate,sdch

HTTP_ACCEPT_LANGUAGE: en-US,en;q=0.8

HTTP_CONNECTION: keep-alive

HTTP_HOST: localhost:5000

HTTP_USER_AGENT: Mozilla/5.0 (X11; U; Linux x86_64; en-US) AppleWebKit/534.0
(KHTML, like Gecko) Chrome/6.0.408.1 Safari/534.0

PATH_INFO: /

...

```
my $app = sub {  
  my $env = shift;  
  return [  
    200,  
    ['Content-Type' => 'text/plain'],  
    ['hello world']  
  ]  
};
```

Réponse

[\$status, \$headers, \$content]

- un code HTTP
- des headers HTTP (sous forme de array ref)
- le contenu de la réponse (de type IO ou un array ref)

De nombreux framework web
sont déjà compatibles

- Catalyst
- Dancer
- Mojo
- Jifty
- Tatsumaki
- Squatting
-

Exemple:

Catalyst

le framework le plus populaire

Abstraction entre le serveur web et Catalyst

Catalyst::Engine::*

(Apache::MP20 CGI FastCGI
Server::Fork HTTP::Prefork, ...)

Dans l'idéal

un seul engine

Catalyst::Engine::PSGI

J'utilise Catalyst,

comment utiliser Plack pour
mon application ?

> cpan Catalyst::Helper::PSGI

Nouvelle application Catalyst ?

```
> catalyst.pl MyNewApp  
created "MyNewAPP"  
created "MyNewAPP/script"  
created "MyNewAPP/lib"  
created "MyNewAPP/root"
```

```
...
```

```
> cd MyNewApp && perl script/mynewapp_create.pl PSGI  
created "MyNewAPP/script/../script/mynewapp.psgi"
```

```
> plackup script/mynewapp.psgi
```

Application Catalyst existante ?

```
> cd MyAPP
```

```
> perl script/mynewapp_create.pl PSGI  
created "MyNewAPP/script/../script/mynewapp.psgi"
```

```
> plackup script/mynewapp.psgi
```

Et si j'utilise Dancer ?

Dancer supporte de base PSGI
!

```
> dancer MyApp
+ ./MyDancer
+ MyDancer/MyDancer.pl
+ MyDancer/app.psgi
+ MyDancer/config.yml
+ MyDancer/environments
...

> cd MyApp && plackup app.psgi
```

Plack

la boîte à outils de PSGI

Plack

Catalyst::Engine::*
(Apache::MP20 CGI
FastCGI Server::Fork
HTTP::Prefor, ...)

Détail des outils suivants :

- plackup
- Plack::Handler
- Plack::Request
- Plack::Builder
- Plack::Middleware
- Plack::Test

Plackup

démarre votre application PSGI en CLI

```
> plackup app.psgi
```

```
HTTP::Server::PSGI: Accepting connections at http://0:  
5000/
```

> `plackup app.psgi -s HANDLER`

sélectionner un serveur web

default => `HTTP::Server::PSGI` (standalone)

```
> plackup app.psgi -E ENV
```

sélectionner un environnement

default => development

Plack::Handler

L'interface entre l'application PSGI et le serveur web

Apache, Apache2, FCGI, Standalone, CGI, ...

Cette couche concerne
(principalement)
les développeurs de framework

PSGI WebServer

- Twiggy
- Starman
- Starlet
- Coronoa

Twiggy

- basé sur AnyEvent
- rapide et léger

Starman

- orienté performance
- prefork
- sockets unix

utilisé en production par mixi
"le facebook japonais"

Support également pour

Apache, FCGI, Nginx, CGI, ...

Apache 2

```
<VirtualHost myapp.example.com>  
ServerName www.myapp.example.com  
ServerAlias myapp.example.com  
DocumentRoot /myapp.example.com
```

```
<Location />  
  SetHandler perl-script  
  PerlHandler Plack::Handler::Apache2  
  PerlSetVar psgi_app /myapp.example.com/app.psgi  
</Location>
```

```
ErrorLog /websites/myapp.example.com/logs/error_log  
CustomLog /websites/myapp.example.com/logs/access_log  
</VirtualHost>
```

FastCGI

```
> plackup -s FCGI \  
  --listen /tmp/fcgi.sock \  
  --daemonize \  
  --nproc 10
```

Plack::Request

Plack::Request

transforme la hash d'environnement PSGI
en objet Plack::Request

Plack::Builder

fournit un DSL pour écrire des middleware

Plack::App::Mount

monter différentes application PSGI

```
my $app1 = sub {  
    200,  
    ['Content-Type' => 'text/html'],  
    ['hello from app1']  
};
```

```
my $app2 = sub {  
    200,  
    ['Content-Type' => 'text/html'],  
    ['hello from app2']  
};
```

```
builder {  
    mount "/app1" => $app1;  
    mount "/app2" => $app2;  
};
```

Plack::Middleware::*

Les middleware sont des
composants réutilisables

Plack::Builder fournit un DSL pour app.psgi

**On peut les voir comme des
Roles dans Moose**

De nombreux middleware

- | | |
|---|--|
| <ul style="list-style-type: none">● AccessLog● Auth● ConditionalGet● ContentMD5● Debug● Session● Throttle● Header● StackTrace● Static● Recursive● RearrangeHeaders● FirePHP● ... | <ul style="list-style-type: none">● Head● Etag● Lint● LogDispatch● ConsoleLogger● Proxy● OAuth● JSconcat● Log4perl● RefererCheck● XFramework● iPhone● Deflater |
|---|--|

```
my $app = sub { [  
    200,  
    [ 'Content-Type' => 'text/html' ],  
    ['hello world']  
] };
```

```
my $middleware = sub {  
    my $env = shift;  
    my $res = $app->($env);  
    $res->[2]->[0] =~ s/world/FPW2010/;  
    return $res;  
};
```

Plack::Middleware::Debug

ajoute des panels de debug à votre
application

```
my $app = sub {  
    200,  
    [ 'Content-Type' => 'text/html' ],  
    [ '<body>Hello World</body>' ]  
};
```

```
builder {  
    enable 'Debug';  
    $app;  
};
```

Plack::Middleware::StackTrace

dump la stack complète en cas d'erreur

```
my $app = sub {  
    die "DIE DIE DIE DIE";  
    [  
        200,  
        [ 'Content-Type' => 'text/html' ],  
        ['hello world'],  
    ];  
};
```

```
builder {  
    enable "StackTrace";  
    $app;  
};
```

Plack::App::Proxy

proxy

```
use Plack::App::Proxy;

builder {
  enable "Proxy::LoadBalancer",
  backends => [
    'http://127.0.0.1:5000',
    'http://127.0.0.1:5001'
  ];
  Plack::App::Proxy->new()->to_app;
};
```

Plack::Test

**plack vient avec les outils nécessaires
pour écrire les tests**

```
use Plack::Test;
use HTTP::Request::Common;

my $app = sub {
  my $env = shift;
  return [
    200,
    ['Content-Type' => 'text/html'],
    ['hello world']
  ]
};

test_psgi app => $app, client => sub {
  my $cb = shift;
  my $req = GET 'http://localhost';
  my $res = $cb->($req);
  ok $res->is_success;
};
```

Questions ?

merci!